Designing a Framework to Handle Context Information

Ana Hristova, Josué Iglesias, Ana M. Bernardos, José R. Casar

Universidad Politécnica de Madrid,
ETSIT, Ciudad Universitaria s/n, 28040, Madrid, Spain
{ana.hristova, josue, abernardos, jramon}@grpss.ssr.upm.es

Abstract. In the recent years, a number of context-aware frameworks have been proposed to facilitate the development of context-aware applications. From the experience gained, in this paper we explore the design principles that context-aware platforms should conform to, the functionalities they have to provide and the technologies and tools that can be used for their implementation. Subsequently, we propose a context-aware framework and describe the architecture it adopts, making our own technological selection from the options previously identified.

Keywords: Context-aware framework, ubiquitous computing, design principles, context-aware applications, middleware, semantic technologies.

1 Introduction

Improving the quality of life and prolonging independent living of elderly people, providing a new tourism experience in a historic site through augmented reality, or managing firefighter brigades depending on their location and availability, are some of the scenarios where the exploitation of the user's situational information (from now on, context information) may be highly beneficial.

To facilitate the design of applications using context information, in the last years, a number of platforms have been developed under different architectural styles, context representation, reasoning capabilities and communication technologies. All of them aim at setting a common practice for building applications and services. Some of these well-known frameworks are Gaia, Context Toolkit, Context Management Framework, SOCAM and CoBrA, which are described in Section 2 below.

In this paper, we explain the general design principles for building a context-aware framework, the functionalities it should possess and the technologies and tools that can be used to implement them (Section 3). In Section 4, we describe a context-aware platform prototype which applies some of the best practices learned and uses some of the considered technologies. Finally, several lines for future work are included in Section 5.

2 A review on context-aware platforms

A number of frameworks have made attempts to set the basis for design and development of context-aware applications. To the best of our knowledge,

architectures based on direct access to sensors, widgets, networked services, blackboards or centralized context servers have been explored to date.

One of the first proposals was the Context Toolkit [1] (2000), which adopts the widget architecture style. The core component in this architecture is the Widget, a software piece that encapsulates the sensor specific access and communication details and notifies subscribed applications when the context changes. Widgets are centrally managed by a component called Discoverer; this component keeps record of all running widgets in the network and is aware of the information provided by each of them. Aggregators and Interpreters are also use to combine and translate context information respectively. The framework adopts the key-value pair context modeling (encoded using XML for transmission) and has no reasoning engine.

SOCAM [2] represents an OSGi-based middleware approach for context-aware services where ontologies are used to convert a physical space into a semantic one. In order to alleviate computation load in mobile devices, it is presented with a two-layer data model. At a lower level there are specific ontologies for particular domains and the generic concepts are modeled above. SOCAM´s architecture is server-based, with several Context Providers acquiring context data and a central Context Interpreter storing and reasoning about context. OWL [21] is used as modeling language and a rule-based reasoning engine is implemented.

Furthermore, COBRA [3] is an agent based architecture that supports context-aware systems in smart spaces. Its central component is the Context Broker, which maintains and manages a shared contextual model on behalf of several Context-Aware Agents. Context is modeled by ontologies represented in OWL and integrated with a rule based inference engine. Also, the architecture includes its own policy language which controls access and enables security and privacy protection.

Two of the main frameworks supporting uncertainty in context-aware systems are Context Management Framework (CMF) [3] and Gaia [4], which adopt the blackboard and distributed middleware architectural design respectively. In CMF a central Context Manager manages the blackboard, processes the context information acquired and infers higher level information. Data acquisition is performed by Resource Servers, while the Context Recognition Services are used on demand by the Context Manager to deduce complex data out of simple context entities. In a similar way, Gaia is intended to coordinate the development and execution of mobile applications for active spaces. Context is modeled and rules are defined by 4-ary predicates, which are supported by several knowledge bases with different reasoning mechanisms. Finally, both projects handle context data in an advanced way, by using ontologies and probability to build higher-level data.

3 Layered functional analysis of a framework to handle context

In this section, we draw conclusions on general design principles that should be taken as guidance when designing a context-aware platform. Also we give an overview of functionalities that the platform should provide and the underlying technologies and tools that can be used for their development.

3.1 General design principles

The design of a context-aware framework should be driven by the design principles detailed below.

- **User-centered design:** The nature of context-aware services requires the frameworks supporting them to comply with the user-centered design principle, where the user does not adapt to the system, but the system transparently adjusts to the user's behavior. Also, the system should not only follow the user's actions and conforms accordingly, but also perform usability analysis to enhance user interaction. It should consider the user's interface preferences and its privacy constraints, therefore positioning the user in the center of its concerns.
- **Layered approach:** By separating the context acquisition from the processing and further on from the application logic, better reusability is achieved. Namely, an application developer can utilize certain sensor information into multiple applications without worrying about low-level sensor details and communication. The layered approach also makes the system easily extendible with new sensor components and easy updatable with new algorithms when the logic needs to be changed without affecting the rest of the system.
- **Easy context retrieval:** The context data should be delivered to the upper layer for further processing and from there sent to the application layer. In order to be more flexible, the architecture should support two basic modes of context retrieval and delivery: 1) *Push*: enables automatic context delivery to all subscribed components; 2) *Pull*: enables on demand context data retrieval. These modes of data delivery should facilitate the application development by allowing several types of subscription mechanism: *continuous, on-demand, periodic and event-based*. The mode of data delivery will depend on the type of context information. For example, an application handling the heart rate may need continuous delivery of heart rate samples to build an electrocardiogram; the most efficient way to provide this information may be through a permanent connection (such as a socket or pipe).
- **Service description and discovery:** Since the sources of context information are not stable and always available ones, keeping an up-to-date registry of the currently available resources, with their general descriptors (in terms of service provider, location, access data and type of delivered information) is a must. With a resource discovery mechanism, the application describes the component it needs by using pre-defined parameters and queries a registry, where the available services are described using common standard languages. In this manner, the application obtains context information of interest or subscribes for receiving automatic notifications from an event dispatcher when changes of context are detected, without being aware of the sensors' details and their implementation and integration in the system. Currently, the industry is imposing convergence to service-oriented architectures.
- **Quality of context:** Context-aware applications' performance is obviously related to the quality of the context information they make decisions on. As a consequence, guaranteeing sufficient context quality in every layer, from acquisition to inference, is needed, in order to compose a sufficiently accurate 'image of context'. This means that error estimates should be calculated from the sensing layer, and merged and updated when necessary, to present context estimates with a valuation of the correctness of the inferred information.

- **Interoperability:** Most of the platforms developed use their own context modeling and handling, exposing this information without having a common service interface. Service Architectures such as Web Services are lately targeting service integration on the Internet and in enterprise architectures, and might provide just the approach needed for interoperability of context-aware frameworks. This would support usage of well established modeling languages and protocols for communication, which successively would increase services' reusability.
- **Scalability:** A context-aware framework should be scalable in terms of maintaining the usual response time as the client number increases and operation load grows. Also, it should include smooth addition of sensors in the system, new algorithms and inferred methods, and new client applications, without disturbing the normal functioning of the system.

3.2 Functionalities and underlying technologies

Next we identify some alternatives to implement acquisition, reasoning and messaging functionalities in a framework to manage context data. Some options to address the design of the global framework are also described.

1. Architecture: Assuming a service-oriented approach, there are a number of technologies that can be chosen as a foundation for building a context-aware framework. Its selection might affect the scalability of the system, the interoperability, the number of users supported, the response time, the reusability and the sensor/service discovery capability. Table 1 provides an overview of several available technologies, considering the advantages and disadvantages that they offer as a basis for context-aware platforms.

Table 1. Technologies for building a context-aware framework.

Technology	Description	Advantages	Disadvantages
OSGi	Open Services Gateway initiative (OSGi) is a dynamic component framework for Java. Applications are composed from a number of reusable components which hide their implementations from one another [10].	- Reusability of components; -Aggregation of context information in the bottom layer; - Easy implementation; - Platform independence: OSGI specifications can be implemented on different types of hardware devices and operating systems because they are based on Java; - Service discovery in own framework; - Non standard Java types are supported by adding an injection lists of data types [5]; - A servlet can be encapsulated in an OSGi bundle.	-Not interoperable with other non-JAVA frameworks; - No service discovery from other OSGi frameworks (work on Remote OSGi is ongoing) [5].
Web Services	A Web service is a software system designed to support machine-to-machine interaction over a network. It uses well	- Supports interoperability by being independent from the language and the platform [7]; - Uses standardized description language: WSDL;	- No automatic aggregation on bottom layer (no listeners between web services); - Need for a client for

	established standards such as: XML, SOAP, WSDL and UDDI [8].	- Dynamic service discovery is enabled by UDDI; - Services can be accessed remotely; - Scalability and better performance on query execution [6].	testing purposes.
Servlets	Servlets are JAVA classes that dynamically process requests and construct responses. The content produced is HTML or XML [9].	- Easy to implement; - Can be invoked from any client, no matter of the implementation.	- Absence of a registry for service discovery; - The client is not aware of the input parameters; - No automatic aggregation of context data on the bottom layer.

2. Messaging (or technologies for APIs): The context-aware framework should be able to deliver data of interest to the applications, having the application to explicitly request for data, or subscribing to the system to receive automatic notifications when new data are obtained. As previously stated, several subscription types can be implemented: on-demand, continuous, periodic and event-based. In order to support these subscription methods and especially the push paradigm, several technologies that could be utilized for handling messaging are reviewed in Table 2.

Table 2. Technologies for handling messaging in a context-aware framework.

Technology	Description	Advantages	Disadvantages
XML/HTTP	One of the simplest ways of messaging: Internet HTTP protocol is used to query web servers, getting back server information in XML format	- Platform-independent and standard – based technologies - Flexible choice of XML tags and structure - HTTP itself offers security mechanisms - XML allows validation using DTD or XML-Schemas - Many tools for HTTP and XML management already developed for almost every platform	- There exist fasters protocols than HTTP - XML itself lacks semantic - No intrinsic data type support
Java Message Service	Java Message Service is a JAVA messaging standard that allows applications to create, send, receive, and read messages [12]	- Asynchronous sending and receiving data -Supports point-to point and publish and subscribe mechanism - Provides different levels of message delivery reliability - Provides standard message formats - Loose coupling - Can be used by non-JMS clients	-Slower communication -Increased network traffic due to sending header and other information together with the message content
Sockets	A socket is an end point of a communication link between two programs in an IP based network	- Easy implementation - Low network traffic - Client and server can be written in different languages	-No standardized message format -Send only packets of raw data, that should further on be handled on the client and server-side
Pushlet	Pushlet is a servlet-based JAVA	-Continuous sending of new HTML content	- The client has to be a web browser

	mechanism that enables pushing content from server-side JAVA objects to a client browser [11]	-Small protocol overhead; -Uses standard HTTP ports and protocols -Light-weight client side (no need for browser plug-ins)	-Scalability: As the number of users increases, the number of resources that are hold increases (threads, sockets etc) -A web server is not designed for long-lived connections -DHTML is not compatible across browsers
Remote Method Invocation	Remote Method Invocation is a distributed Java technology that enables executing computations remotely [13]	- Handles threads and sockets - Its communication falls back to sockets and if restricted by a firewall, falls back into HTTP communication -Supports distributed architectures for context acquisition and reasoning, as it allows communicating different virtual machines	-Uses non standard ports that might be blocked by firewalls -Requires a dedicated server -It can be only used by JAVA clients -Vulnerable security -Tightly coupled. Requires that the client has a pre-knowledge of the methods to be called - It is supported in few browsers
SOAP **(Simple Object Access Protocol)**	SOAP provides mechanisms for exchanging structured and typed information between peers in a decentralized, distributed environment [14]	- Simple and extensible - Platform and language independent - Based on XML -Transport protocol independent (HTTP, SMTP, etc.) - SOAP allows you to get around firewalls	- SOAP contains no mention of security facilities - SOAP clients do not hold any stateful reference to remote objects

3. Context modeling: Knowledge representation is one of the most important areas in the field of context-sensitive systems. Such systems must integrate information from several heterogeneous sources. In this way, the data flow between different components requires the definition of a model describing information in a structured way so that it can be stored, computed, shared and reused.

Following the Linnhof-Strang classification [15], the *key-value model* is the simplest approach: very easy to manage but not offering algorithms to support efficient access to information; it is therefore not suitable for highly dynamic contextual applications. *Mark-up schemes* models are hierarchical tag-based models with attributes and content, usually derived from SGML (as the well-known XML). There are several proposals to model contextual information with mark-up schemes (CSCP, CC/PP Context Extension, PPDL, etc.). Context has also been modeled using UML diagrams, the most important of the so-called *graphical models*. These have mainly been used to describe the structure of the contextual knowledge base. *Object-oriented modeling techniques* can be also used to model context in order to exploit reusability, inheritance, etc. These models are suitable for distributed environments, typical of the ubiquitous computing, although they may lose capacity sharing concepts as a result of encapsulation. *Logic-based models* have a high degree of formalism that can be applied to infer new knowledge. Finally, *ontologies* provide a shared vocabulary used to model a domain, that is, the type of objects and concepts that exist, and their properties and relations. Ontology-based models combine the advantages of object-oriented and logic-based ones, adding semantic and maintaining formalisms that allow validating partially the model and inferring new information.

This last approach is the most popular nowadays. Among the technologies which facilitate the use of ontologies, OWL, Protégé and Jena are the useful tools. The first one is a semantic language for publishing and sharing ontologies on the Web [21]. Protégé is an open source editor for modeling ontologies, which alleviates creation, visualization and manipulation of ontologies in different formats [22]. Finally, Jena is an API for programmatically managing ontology models from JAVA programs [23].

A particular issue when facing context modeling is how to proceed with sensors. Sensors are sources of context data and depending on the way the data is provided, it may be needed to cope with physical, logical and virtual sensors. To model sensors, Sensor Model Language (a model for discovering, querying and controlling web-resident sensors [17]) may be used. Other alternative is the sensor database system (stores the sensors' characteristic and data in a database, to which it communicates through an Abstract Data Type interface [15]) etc.

Table 3. Some of the most used data models to represent context information.

Approach	Advantages	Disadvantages
Key-value	- Low complexity; easy to manage - Applicability to existing ubiquitous computing environments	- Lack of formality - Do not offer algorithms to support efficient access to information - No partial validation is possible - Difficult to add quality meta-information to contextual information
Mark-up Schemes	- They can be partially validated - Comprehensive set of validation and management tools do exist - Quality meta-information may be added to contextual information - Applicability to existing infrastructure based on marking, such as Web Services	- Incompleteness and ambiguity have to be handled proprietary on the application level - Not fully scalable (hierarchical structure but not inherence)
Ontologies	-Structured, explicit and formal description of concepts and the relations between them - Highly scalable (inheritance support) - Knowledge can be easily shared and reused - Can be partially validated - Comprehensive set of validation tools exist - Quality meta-information may be added to contextual information - Able to infer new information. - Can be integrated with other kinds of reasoning methods [18]	- Development complexity is high - Reasoning complexity is high

4. Reasoning: A challenge in context-aware computing is inferring rich information from low-level context data. For example, *rule-based reasoning* is the simplest reasoning approach; it is widely used to complement other types of reasoning. Specifically, it fits seamlessly with ontological models. *Fuzzy logic* uses expressions that are neither entirely true nor completely false, being able to represent common knowledge in a mathematical language through set theory. *Bayesian Networks* are graphical models that combine probability and graph theory to represent several related uncertainties, providing an excellent tool to manage complex problems of uncertainty. *Case Based Reasoning* is concerned with solving new problems by remembering similar earlier experienced situations. Having a knowledge base of

solved problems, this kind of mechanism applies four sequential phases: retrieve, reuse, revise and retain. *Dempster-Shafer inference* is used to combine separate pieces of information (evidence) to calculate the probability of an event. In the following table an outline of some of these reasoning approaches is given, together with the tools that can be used to implement reasoning procedures.

Table 4. Reasoning approaches to infer complex context information.

Approach	Advantages/Disadvantages	Tools
Rule-based reasoning	+ New rules can be added in a simple, incremental fashion, without rewriting the whole system + Knowledge is represented in an explicit and intelligible way + Easy to integrate with other reasoning methods (as ontologies) - Contextual information can't be effectively managed - Automatic rules acquisition is difficult	- Bossam: inference engine with native support for reasoning over OWL ontologies and SWRL/RuleML rules [19] - Jess: small, light and fast rule engine for the Java platform where rules can be defined in XML [20] - Other tools: Pellet, FaCT, RacerPro, etc.
Bayesian Networks	+ Are based on probability theory, a consistent mathematical tool + Can readily handle incomplete data sets + Able to build complex systems from simpler ones + The belief is automatically kept updating when new data item arrives + Learning about causal relationships - Bayesian networks need to be trained - Highly dependent on prior knowledge reliability - No universally accepted method for constructing a network from data - Often not practical for large systems	- Bayes Net Toolbox: open source Matlab package for directed graphical models that supports probability distributions, inference and structure learning [26]. - MSBNx (Microsoft Bayesian Network Editor): a component-based application for creating, assessing, and evaluating Bayesian Networks [25]. - WEKA: a collection of machine learning algorithms for data mining tasks [24]. - Other tools: BNJ, JavaBayes, BayesiaLab, BUGS, etc [27].
Fuzzy Logic	+ Mimic human decision-making to handle vague concepts + Ability to deal with imprecise or imperfect information + Improved knowledge representation and uncertainty reasoning - Highly abstract and heuristic	- Matlab's Fuzzy Logic Toolbox: extends the Matlab computing environment with tools for designing systems based on fuzzy logic [28]. - Other tools: FuzzyCOPE, fuzzyTECH, etc [29].

4 Design of a context-aware framework for acquisition purposes

After reviewing the general design principles in the context-aware application domain, in this section we propose an architecture that addresses some of the issues analyzed before (Fig. 1). In particular, it decouples context acquisition from context reasoning, by providing standardized APIs to sensors information. The inference logic is to be included in upper levels. After the review of the available technologies, the acquisition middleware has been implemented by using servlets and application interfaces are offered through XML/HTTP messaging.

Sensors constitute the bottom layer and they either operate independently one from another, or are connected in a sensor network, whose data is gathered by a single

sensor server (e.g. it is the case of ZigBee wireless sensor networks). Furthermore, servlets have been used as the technology that enables the interaction of the platform with its environment; the servlets' container shape a context provider, encapsulating the underlying sensor information and enabling applications to access this information remotely. Communicating with the sensor servers or with the sensors themselves, the servlets process and structure the data obtained and deliver it to the clients usually in XML format.

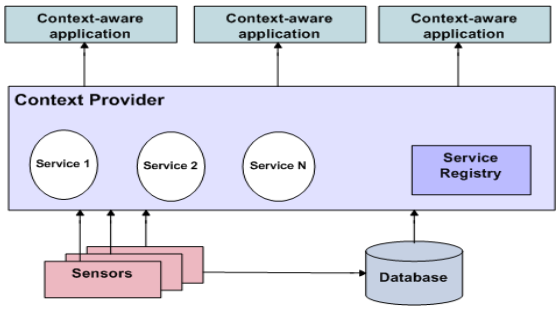

Fig. 1. Overview of the architecture of the context-aware framework.

For the communication between distributed objects, sockets have been used, because they do not demand language compatibility between client and server and are easy to implement. Finally, the application is able to discover the available services by querying a service registry, which contains a list of the available services. The service registry has been implemented using the servlet approach and integrates a query mechanism that returns an XML description of the services of interest containing: the servlet name, the input parameters, the sensors, the type of notification, and the format of the output result. The elements of which the system's infrastructure is composed of are shown in Fig.2.

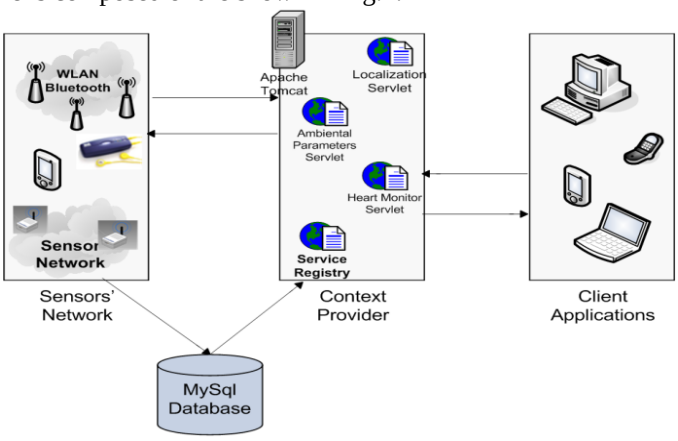

Fig. 2. Overview of the context-aware infrastructure.

In this manner, a loosely coupled layered architecture is achieved which supports the interoperability requirement by being accessible from any platform,

without specific demands regarding the application implementation, and by using standardized language such as XML for services and context data description.

5 Conclusion and future work

In spite of the various proposals that have been presented to date, there is not an agreed methodology to cope with context information and easily develop new applications using context data. For that reason, in this paper we have reviewed the general design principles that need to be followed when designing a context-aware framework and have outlined some of the possible underlying technologies that can be chosen to implement an essential functionality in the system. Moreover, we have described our implementation of a context acquisition system, by choosing some of the technologies analyzed earlier, and the interactions between its components.

The platform developed does not drastically differ from the others analyzed. It follows the layered approach, addresses the main points of concern and is designed in a way to be easily extendible by addition of other components that should comprise a complete context-aware platform. From the experienced gained, a hurdle was recognized in the variety of sensors, their specifications and interfaces which need to be studied separately in order to extract the context information of interest.

As further work we intend to give a detailed specification of all the fundamental elements of the framework and their correlation as well as to introduce redundancy in order to increase the reliability of the system. Quality of context is also a pending issue. Also, we tend to focus on approaches for distributed context acquisition and management, and review the requirements for developing a lightweight context management system for mobile devices.

Acknowledgments. This work has been supported by the Government of Madrid under grant S-0505/TIC-0255 and by the Spanish Ministry of Science and Innovation under grant TIN2008-06742-C02-01.

References

[1] Dey, A.K., Abowd, G.D., Salber, D.: A Conceptual Framework and a Toolkit for Supporting the rapid Prototyping of Context-Aware Applications. In Human-Computer Interaction Journal, Vol. 16 (2-4), 2001, pp. 97-166.

[2] Gu, T., Pung, H.K., Zhang, D.Q.: Toward an OSGi-based infrastructure for context-aware applications. In IEEE Pervasive Computing 10-12, 2004, pp. 66-74.

[3] Singh, A., Conway, M.: Survey of Context aware Frameworks - Analysis and Criticism, In UNC-Chapel Hill ITS, The University of North Carolina, 2006.

[4] Roman, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H., Nahrstedt, K.: Gaia: a middleware infrastructure to enable active spaces. In IEEE Pervasive Computing 10-12, 2002, pp.74-83.

[5] Rellermeyer, J. S., Alonso, G., Roscoe, T.: R-OSGi: Distributed Applications through Software Modularization. In Proceedings of the ACM/IFIP/USENIX 8th International Middleware Conference, 2007.
[6] De Almeida, D. R., de Souza Baptista, C., da Silva, E. R., Campelo, C. E. C., de Figueirêdo, H. F., Lacerda, Y. A : A Context-Aware System Based on Service-Oriented Architecture. In 20th International Conference on Advanced Information Networking and Applications, Volume 1, 18-20 April 2006, pp. 205-120, doi10.1109/AINA.2006.16.
[7] Chaari, T., Laforest, F., Celentano, A.: Service-Oriented Context-Aware Application Design. In First International Workshop on Managing Context Information in Mobile and Pervasive Environments, Ayia Napa, CYPRUS, 2005.
[8] Web Services Architecture, http://www.w3.org/TR/ws-arch/wsa.pdf
[9] Sun Home Page – Java Servlet Technology, http://java.sun.com/products/servlet/
[10] OSGi- The Dynamic Module System for Java, http://www.osgi.org/Main/HomePage
[11] Sourceforge Home Page - Pushlets, http://sourceforge.net/projects/pushlets
[12] Sun Home Page - Java Message Service, http://java.sun.com/products/jms/
[13] Sun Home Page - Remote Method Invocation, http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp
[14] Simple Object Access Protocol (SOAP) 1.2 - http://www.w3.org/TR/soap/
[15] Strang T., Linnhoff-Popien C. A Context Modeling Survey. First International Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp. 2004
[16] Bonnet, P., Gehrke, J., Seshadri, P.: Towards Sensor Database Systems. In Proceedings of the Second International Conference on Mobile Data Management, p.3-14, January 08-10, 2001, Springer-Verlag, UK.
[17] Sensors Online Magazine, http://www.sensorsmag.com/articles/0403/30/main.shtml
[18] Studer, R., Benjamins, R. Knowledge Engineering: Principles and Methods. Data and Knowledge Engineering, 25(1-2), pp. 161-197. 1998
[19] M. Jang y J. Sohn, “Bossam: An Extended Rule Engine for OWL Inferencing,” Rules and Rule Markup Languages for the Semantic Web, 2004, pp. 128-138.
[20] Jess Rule Engine - http://www.jessrules.com/
[21] OWL, Web Ontology Language, Overview - W3C Recommendation. 10, February 2004 http://www.w3.org/TR/owl-features/
[22] Protégé, http://protege.stanford.edu/
[23] Jena, http://jena.sourceforge.net/
[24] Bouckaert, R.R. Bayesian Network Classifiers in Weka. 2005
[25] Kadie, C.M., Hovel, D. MSBNx: A Component-Centric Toolkit for Modeling and Inference with Bayesian Networks. Microsoft Research Technical Report MSR-TR-2001-67, July. 2001
[26] Murphy, K.P. The Bayes Net Toolbox for Matlab. Computing Science and Statistics, vol. 33, 2001, pág. 2001
[27] Murphy, M. Software Packages for Graphical Models / Bayesian Networks, http://www.cs.ubc.ca/~murphyk/Software/bnsoft.html
[28] Matlab’s Fuzzy Logic Toolbox, http://www.mathworks.com/products/fuzzylogic/
[29] Fuzzy Logic Software, http://www.cs.cofc.edu/~manaris/ai-education-repository/fuzzy-tools.html
[30] Neapolitan, R.E. Learning Bayesian Networks, Prentice Hall, 2003.